WTP2025

SERVERLESS ARCHITECTURE

API Gateway-Lambda-DynamoDB

ABOUT
API GATEWAY

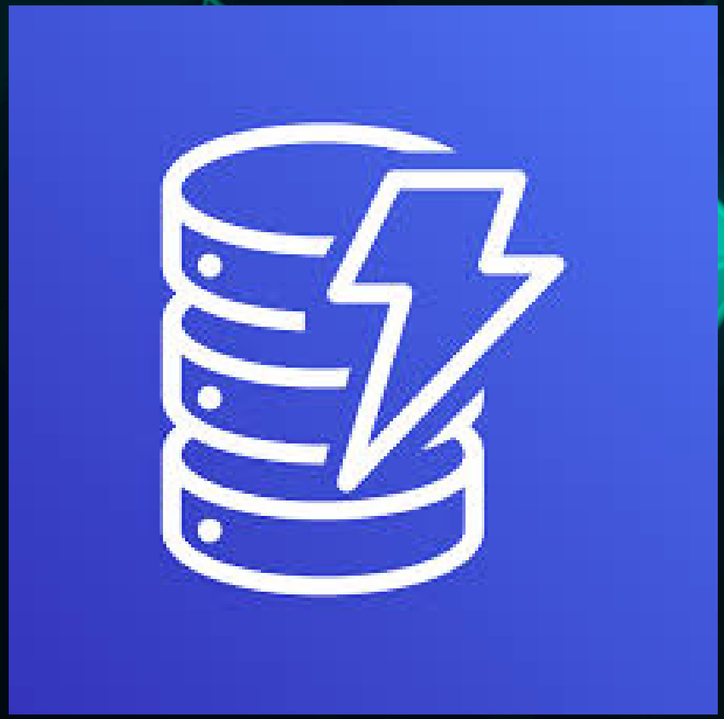
1. 완전 서버리스 → 서버 관리 불필요, 자동 유지보수
2. 고성능 & 확장성 → 대량 요청 처리, 자동 확장 지원
3. 비용 효율성 → 사용량 기반 요금제, 캐싱으로 비용 절감
4. 강력한 보안 → IAM, Cognito, JWT 인증 지원
5. 다양한 API 지원 → REST, GraphQL, WebSocket API 제공
6. 트래픽 제어 → 레이트 리미팅, 캐싱, 모니터링 가능
7. 서버리스 서비스와 최적 조합 → Lambda, DynamoDB, S3 등과 원활한 연동

ABOUT LAMBDA

1. 완전 서버리스 → 서버 관리 없이 코드 실행 가능
2. 자동 확장 → 트래픽 증가 시 인스턴스 자동 생성
3. 비용 효율성 → 사용한 만큼만 비용 지불 (요청당 과금)
4. 고성능 → 병렬 실행 지원, 이벤트 기반 빠른 응답
5. 다양한 언어 지원 → Python, Node.js, Java 등 사용 가능
6. 광범위한 AWS 서비스 연동 → API Gateway, DynamoDB, S3 등과 최적화
7. 보안 강화 → IAM 역할 기반 액세스 제어 적용 가능

ABOUT DYNAMODB

1. 서버리스(Serverless)
    - 서버 프로비저닝, 소프트웨어 관리, 유지보수 불필요
    - 가동 중지 없이 유지 관리 가능
2. 온디맨드 용량 모드
    - 사용량 기반 요금 부과 (사용한 만큼 지불)
    - 자동으로 스케일 업/다운
    - 트래픽이 없을 경우 비용 발생 없음
3. NoSQL 데이터베이스
    - 키-값 및 문서 데이터 모델 지원
    - JOIN 연산 미지원 → 비정규화 데이터 모델 권장
4. 완전관리형(Managed Service)
    - 설정, 유지보수, 보안, 백업, 모니터링 자동 처리
    - 가용성 및 성능 지속적 개선 (업그레이드 불필요)
5. 고성능 및 확장성
    - 규모에 관계없이 10ms 미만 응답 속도 제공
    - JOIN 연산 등 비효율적인 기능 제외 → 성능 최적화
    - 수백 명~수억 명 사용자가 있어도 성능 유지



API GATEWAY-LAMBDA-DYNAMODB

- 완전 서버리스*(Serverless):* 서버 관리 불필요, 자동 확장, 유지보수 부담 최소화
- 비용 효율성*:* 사용한 만큼만 비용 지불, 유휴 상태 시 비용 절감
- 고성능 및 확장성
  - *API Gateway:* 수백만 개 요청 처리 가능
  - *Lambda:* 자동 스케일링으로 병렬 처리 지원
  - *DynamoDB: 10ms* 미만의 빠른 응답 속도 및 무제한 확장
- 운영 및 유지보수 최소화*: AWS*에서 인프라 자동 관리, 패치 및 업데이트 불필요
- 보안 강화*: IAM, Cognito, JWT* 등으로 강력한 인증 및 권한 관리
- 유연한 개발 가능*:* 다양한 프로그래밍 언어 지원, 마이크로서비스 및 이벤트 기반 아키텍처 구축 가능
- 다양한 *API* 지원*: REST API, GraphQL API, WebSocket API* 구현 가능

TODAY'S GOAL

*API Gateway - Lambda - DynamoDB*를 활용한 완전 서버리스
*CRUD API*를 구축하고 데이터 베이스 구조를 이해*!*

TODAY'S GOAL

1 기본 환경 설정
- *API Gateway, Lambda, DynamoDB* 생성 및 연결
- *AWS IAM* 역할 설정 및 필요한 권한 부여

2 *CRUD* 기능 구현 및 테스트
- *Create (생성): API Gateway*를 통해 데이터를 *DynamoDB*에 추가하는 *Lambda* 함수 구현
- *Read (조회):* 특정 항목을 조회하는 *GET API* 구현
- *Update (수정):* 기존 데이터를 업데이트하는 *PUT API* 구현
- *Delete (삭제):* 특정 항목을 삭제하는 *DELETE API* 구현

3 테스트 및 검증
- *Postman* 또는 *cURL*을 활용하여 *API* 호출 및 응답 확인
- *DynamoDB*에 데이터가 정상적으로 반영되는지 확인

4 성능 및 확장성 고려
- *API Gateway*와 *Lambda*의 응답 속도 확인
- *DynamoDB*의 온디맨드/프로비저닝 모드 차이 이해

CORS (CROSS-ORIGIN RESOURCE SHARING)란?

- CORS (교차 출처 리소스 공유)는 웹 브라우저가 다른 도메인(ORIGIN)의 리소스 요청을 허용 또는 차단하는 보안 정책입니다.

◆ 왜 CORS가 필요한가? (SAME-ORIGIN POLICY)

- 기본적으로 웹 브라우저는 보안상의 이유로 동일 출처(SAME-ORIGIN)에서만 리소스를 주고받을 수 있도록 제한합니다.
- 즉, HTTPS://EXAMPLE.COM에서 로드된 웹 페이지는 HTTPS://API.EXAMPLE.COM처럼 다른 도메인의 API에 요청을 보낼 수 없습니다.
- 하지만, API 서버가 CORS 정책을 설정하면 다른 도메인에서도 API를 사용할 수 있도록 허용할 수 있습니다.

◆ CORS 작동 방식

- PREFLIGHT REQUEST (사전 요청)
- 클라이언트가 API 서버에 OPTIONS 메서드로 요청을 보내 CORS 허용 여부 확인
- 서버가 허용하면 브라우저가 실제 요청을 보냄

- CORS RESPONSE HEADERS (서버에서 허용 여부 설정)
- ACCESS-CONTROL-ALLOW-ORIGIN: 허용할 도메인 지정 (*은 모든 도메인 허용)
- ACCESS-CONTROL-ALLOW-METHODS: 허용할 HTTP 메서드 (GET, POST, PUT, DELETE)
- ACCESS-CONTROL-ALLOW-HEADERS: 허용할 요청 헤더 (CONTENT-TYPE, AUTHORIZATION 등)

WTP2025

THANK YOU!

FOR YOUR ATTENTION